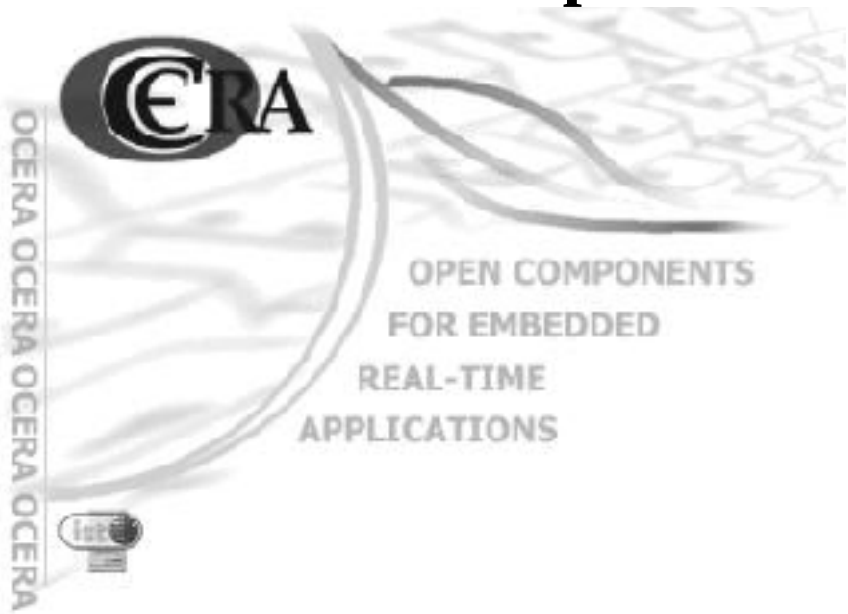
WP9 Validation on platform



Deliverable D9pc.2
Process control application component specification

WP9 Validation on platform : Deliverable D9pc.2 Process control application - component specification
by Ales Hajny and Petr Cvachoucek

Published May 2003

Table of Contents

Chapter 1. Description of control system software structure 1
Description of control algorithm interpreter ExeCont.....1
Chapter 2. Component Specification.....4
Developed components.....4
Planned use of OCERA components.....4
Description of components.....5
Interface TioStore (technology IO store).....5
Interface MsgStore (message store).....6
Interface IpcMsgQueue (interprocess message queue).....7
Process NdMan (node manager).....8
CanIO subsystem.....8
Ethernet communication.....9

Document Presentation

Project Coordinator

Organisation:	UPVLC
Responsible person:	Alfons Crespo
Address:	Camino Vera, 14, 46022 Valencia, Spain
Phone:	+34 963877576
Fax:	+34 963877576
Email:	alfons@disca.upv.es

Participant List

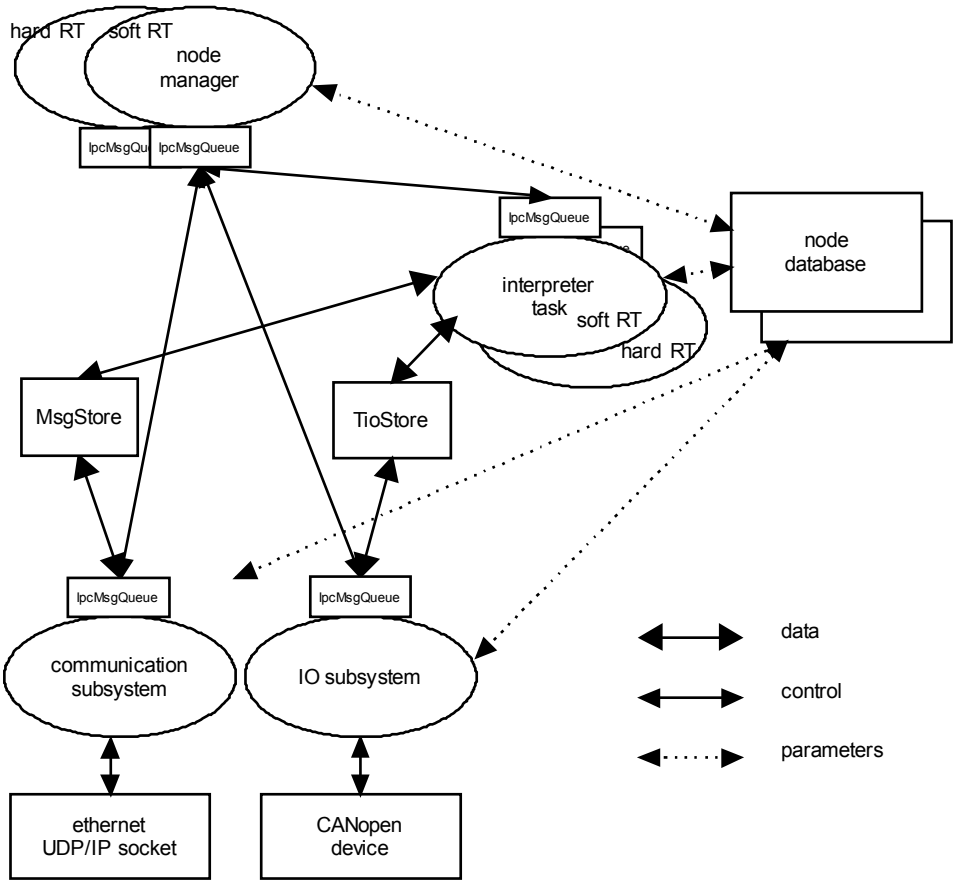
Role	Id.	Participant Name	Acronym	Country
CO	1	Universidad Politecnica de Valencia	UPVLC	E
CR	2	Scuola Superiore Santa Anna	SSSA	I
CR	3	Czech Technical University in Prague	CTU	CZ
CR	4	CEA/DRT/LIST/DTSI	CEA	FR
CR	5	Unicontrols	UC	CZ
CR	6	MNIS	MNIS	FR
CR	7	Visual Tools S.A.	VT	E

Document version

Release	Date	Reason of change
1_0	27/05/03	First release

Chapter 1. Description of control system software structure

Control system SW consists of blocks as depicted in the following figure. The blocks exchange information via data IO interface. The control system kernel is an interpreter of control algorithms which meets the requirements of ISO 61131. The other system modules ensuring control system node management, communication between nodes and data transmission from process interfaces will be connected to the interpreter.



Description of control algorithm interpreter ExeCont

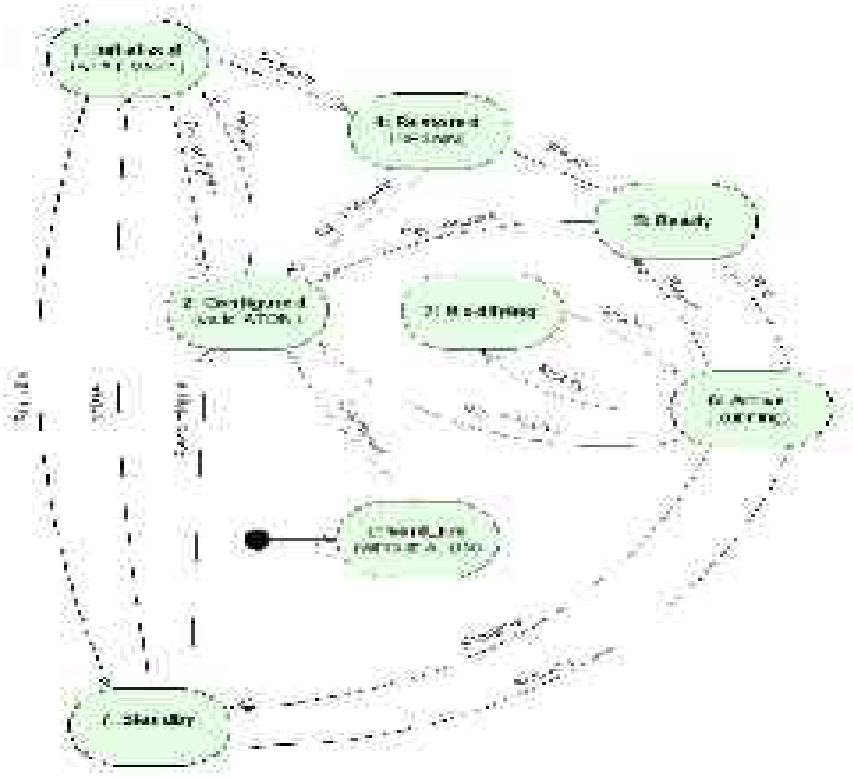
Application ExeCont includes the following program modules:

- MessProc - module for processing messages sent by other stations or processes and received via IpcMsgQueue interface, generation of replies and their sending out via IpcMsgQueue.
- MessGen - module for generation of events on application program variables, generation of messages to other stations or processes and their sending out via MsgStore.

- Modules of libraries of execution functions of FWE object interpretation classes
- KerExe - Interpreter kernel module

  The interpreter kernel contains three-level hierarchy of C++ objects, and an additional fourth level from the viewpoint of UniCAP objects (order from the top of the hierarchy downwards):

  1. The only object of cNdExe class covering the whole interpreter
  2. Up to three subordinated objects of cTaskPc class reprezenting individual priority classes of category TASK
  3. Objects of cTask class corresponding to the objects of category TASK defined for given node in UniCAP
  4. FWE objects



State diagram of interpreter UCtasks of ATDN node target database

cTask function:Exec() forms the interpreter kernel, and includes three gradually executed parts:

1. Preparation of data entering the task - potential initialization of variables, download of data addressed to the task by the tasks of another nodes (via MsgStore interface), download of input data from local and remote IO modules (via TioStore interface) and their eventual sending for transmission via synchronization channel into the node in Standby state.
2. Interpretation loop, in which interpretation functions FWExec are gradually activated for individual instances of FWE objects.
3. Output variable data transfer for transmission to IO modules (via TioStore interface), generation of events and event messages with transfer for sending the messages for other processes and tasks of other nodes (via MsgStore interface), and potential generation of synchronization messages and their transfer for transmission into Standby node.

The interpretation loop is written as an endless one, and its quitting is realized by a mechanism of user exception enforced in the body of an object of End type, or as a consequence of a run-time error.

Chapter 2. Component Specification

Developed components

To connect the interpreter to the control system node, the following data IO interfaces and control system modules will be implemented in OCERA environment:

- Interface TioStore (technology IO store)
- Interface MsgStore (message store)
- Interface IpcMsgQueue (interprocess message queue)
- Node manager - process NdMan
- CanOpen IO subsystem – CanIO
- Ethernet communication – process ExeCom

Planned use of OCERA components

- CanOpen - DS301
- CanOpen analyzer
- Ethernet communication – RTPS
- Semaphore between user and kernel space
- Shared memory between user and kernel space
- Signals between user and kernel space
- PowerPC implementation

Description of components

Interface TioStore (technology IO store)

Interface TioStore will serve for handing over of values of variables among more processes. It will enable store of variables of various data types, and offer tools for synchronization of processes in access to variables stored in the interface. Internal implementation and structure of the interface will be fully hidden for the applying processes; they will work with the interface by means of a set of methods (API).

Interface requirements:

- Interface contains a set of slots - slot is an array of variables of identical data type
- Slots are identified by their ID (ordinal number) and a short name in the interface
- Slot contains, besides an array of variables, also state - an item to be freely used by the processes connected to the interface (handing over state information such as validity of variable values etc.)
- A number of processes can be connected to the interface simultaneously, the maximum number of the processes connected is specified as a parameter in the interface design
- Access to the variables in the interface is synchronized by means of a lock (mutex); every elementary operation (read, write of variables) is lockable
- Calling processes can lock the interface for the whole sequence of individual elementary operations
- The number of connected processes, their PIDs and names can be determined by means of API interface
- Processes connected to the interface can send signals to each other and thus control the access to variables (the process which modifies the variables in the interface can send this information to the other processes connected)
- For every connected process, the interface contains a memory (queue) of several recent entries - it is designed for the processes which want to process only changes in the interface; the number of memorized changes is a parameter set in the interface design - the function is only executed, if a connected process requires this functionality
- The interface is implemented as a C++ object
- Implementation uses objects DataModule (shared memory between processes) and Mutex (lock) from the library UcFramework, which ensure independence on the target platform
- Implementation for soft, hard RT environment and between the both environments

Diagnostic utility DumpTio:

- designed for viewing and modifying the content of TioStore interface
- has a form of a line program controlled from the command line

Components used: shared memory, semaphore and signal between soft and hard RT environment.

Interface MsgStore (message store)

Interface MsgStore will serve for handing over messages (blocks of data of various size) among more processes. The order of read of messages from the interface can be arbitrary (it is selected by the process/processes which takes/take the messages from the interface). The inserted messages are identified by additional tags such as message priority, sender, recipient, user context etc., which are used by the other processes in taking messages from the interface. Internal implementation and structure of the interface will be fully hidden for the applying processes; they will work with the interface by means of a set of methods (API).

Interface requirements:

- Messages are stored in a set of blocks - block is a basic allocation unit in the interface
- Messages longer than one block are divided into more blocks; the interface should make it possible to set whether the message can be fragmented or must lie in a continuous area
- Messages are unambiguously identified by their ID (determined by the number of the block they are stored in)
- A number of processes can be connected to the interface simultaneously, the maximum number of the processes connected is specified as a parameter in the interface design
- Access to the messages in the interface is synchronized by means of a lock (mutex); every elementary operation (message read, write) is lockable
- Calling processes can lock the interface for the whole sequence of individual elementary operations
- The number of connected processes, their PIDs and names can be determined by means of API interface
- Processes connected to the interface can send signals to each other and thus synchronize each other
- Message can have assigned a specification of the recipient, priority and context (user defined information) and a signal (process which wrote the message in the interface can apply for signal sending at the moment when the message is read from the interface)
- Processes connected to the interface can, using its API, also determine what messages are stored in the interface, and select their own algorithms in message selection
- Most common requirements (such as selection of the oldest message in the interface, selection of a message with the highest priority etc.) will be implemented as methods in API interface
- The interface is implemented as a C++ object
- Implementation uses objects DataModule (shared memory between processes) and Mutex (lock) from the library UcFramework, which ensure independence on the target platform
- Implementation for soft, hard RT environment and between the both environments

Diagnostic utility DumpMsg:

- intended for viewing and modifying the content of MsgStore interface
- has a form of a line program controlled from the command line

Components used: shared memory, semaphore and signal between soft and hard RT environment.

Interface IpcMsgQueue (interprocess message queue)

Interface IpcMsgQueue will serve for transfer of messages between processes, when the messages will be read from the interface in the order of their writing (FIFO). The interface provides a simple but very efficient mechanism of interprocess communication. It is designed for one-way transmission of messages from more processes to a queue owner.

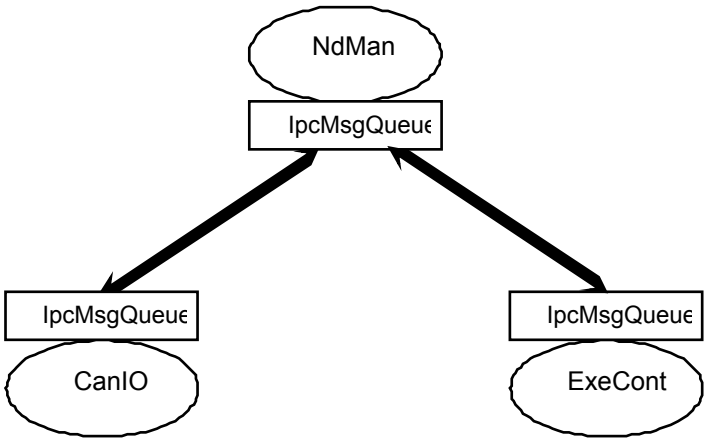
Interface requirements:

- Messages will be stored in a circular buffer in the shared memory (DataModule)
- Buffer access is synchronized by means of a lock (mutex)
- Process writing a message into the queue can call for sending a signal at the moment of message collection from the queue
- Queue owner (the process which created the queue and the only process which read the messages from the queue) can call for sending of the signal at the moment of writing the message into the queue
- Implementation for soft, hard RT environment and between the both environments

Components used: shared memory, semaphore and signal between soft and hard RT environment.

Process NdMan (node manager)

Links between processes and interfaces used:



Requirements for NdMan process:

- At node start, it determines the state of the node itself based on communication with NdMan of the standby node
- It monitors the standby node state, and decides on the state of the node itself (ACTIVE/STANDBY)
- It receives messages from interpreter Execont - requests for a change of the node state coming from the tuning tool UniTun or from internal diagnostics of Execont process
- Monitoring is carried out by means of messages transmitted between NdMan processes in both nodes
- In changes of the state of its own node, it commands other processes (Execont, BackupIO, CanIO)
- Commanding is executed by means of sending messages via IpcMsgQueue interface

CanIO subsystem

Requirements for CanIO subsystem:

- It ensures reconfiguration of CAN subsystem (concurrent copman process) according to the commands from NdMan process
- CAN subsystem is controlled according to current state of the node itself (ACTIVE/STANDBY) in such a way that CAN network is controlled from the active node only

Components used: CanOpen

Ethernet communication

It enables data exchange between control system nodes and control application development station.
Requirements:

- Transmission of data to the target node
- Receipt of data from the target node
- Acknowledgement of data receipt to the source node
- User information on node unavailability
- Dynamic configuration of the network by means of broadcast of node state messages

Components used: RT ethernet